logistic regression as a Neural Network

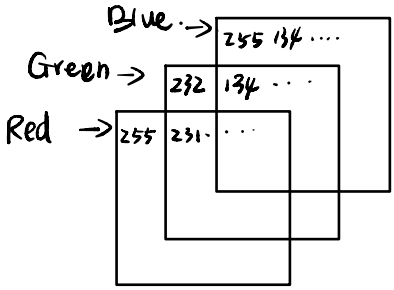
- Problem setting:

logistic regression is a type of binary classification.

sample problem: classify image as 1 (cat) or 2 (non cat)

image size: 64 x 64, with 3 channels.

we usually layout each single input training data as a flattened 1-D array.



in this case: $x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 232 \\ 134 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$ $n = n_x = 64 \times 64 \times 3 = 12288$.

our goal (model): $x \rightarrow y$; Given any x ($n_x = 12288$), predict y (1 or 0, stands for cat or non-cat).

- Notation.

m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots (x^{(m)}, y^{(m)})\}$

a single training example: (x, y), $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

$m = M_{train}$ = # of trainning data, M_{test} = # test example.

2 ways to represent training data / test data:

① $X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$ $\uparrow$ n_x (size of input data) $\downarrow$

② $\begin{bmatrix} — & x^{(1)T} & — \\ — & x^{(2)T} & — \\ & \vdots & \\ — & x^{(m)T} & — \end{bmatrix}$

In order to reach the full power of neural network, we would normally use the first representation for data (input, output, train, test). More specifically, all data will be treated as column vectors.

In the previous example: $X \in \mathbb{R}^{n \times m}$

$$X.shape = (n_x, m).$$

$$Y = [\, y^{(1)}, y^{(2)} \dots y^{(m)} \,]$$

$$Y \in \mathbb{R}^{1 \times m}$$

$$Y.shape = (1, m)$$

Formal Definition of Logistic Regression.

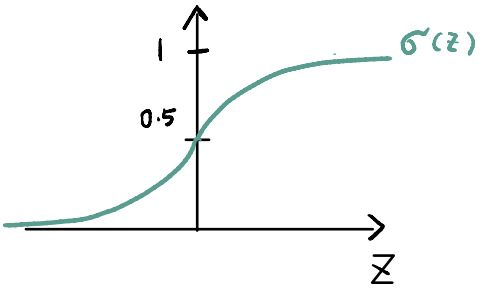
Given x, input, want $\hat{y} = P(y=1 \mid x) \in (0,1)$.
output

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

$$\Rightarrow \hat{y} = w^T x + b$$

we need $\hat{y}$ to be a number between 0 and 1.

$$\Rightarrow \hat{y} = \sigma(w^T x + b)$$



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

if z large, $\sigma(z) \approx \frac{1}{1+0} = 1$

if z small (large negative)

$$\sigma(z) = \frac{1}{1+\infty} = 0$$

lost function and cost function:

problem: $\hat{y} = \sigma(w^T x + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

Given $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$

We need a loss function to quantify how well does the model predict on the input data.

loss function: high loss, bad prediction.
low loss, good prediction.

Any function of $l(\hat{y}, y)$, can be loss functions, However, some loss functions are not suitable for training neural networks.

eg. $\frac{1}{2}(y-\hat{y})^2$: this is not a convex function, Thus it has a lot of local mins, making it hard or impossible to work well with gradient descent.

Example of good loss function:

$l(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

if $y=1$, $l(\hat{y}, y) = -\log \hat{y}$, $\hat{y}\uparrow$, $l(\hat{y}, y)\downarrow$ ✓

if $y=0$, $l(\hat{y}, y) = -\log(1-\hat{y})$, $\hat{y}\downarrow$ $l(\hat{y}, y)\downarrow$ ✓

Cost function:

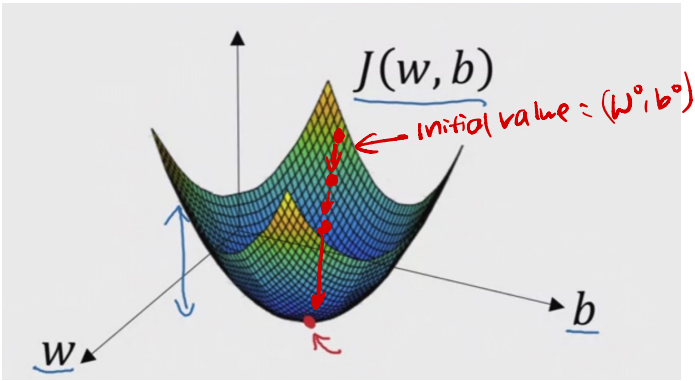
$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{l(\hat{y}^{(i)}, y^{(i)})}_{\text{loss function.}}$$

Cost function gets cost of my parameters, in training, we wish to find the parameters w and b that minimizes cost function.

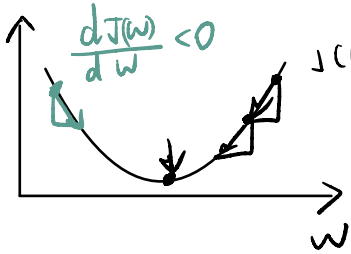
Gradient Descent:

This is the technique that gradually adjusts training parameters and brings cost function to a local minimum.

Cost function for logistic regression looks like below.



Simpler case: ignoring b for now.



Repeat {

$$w := w - \alpha \frac{dJ(w)}{dw}$$

Learning rate

Gradient of cost function at the moment.

Back to logistic regression:

$J(w, b)$

$$w := w - \alpha \frac{dJ(w,b)}{dw}$$

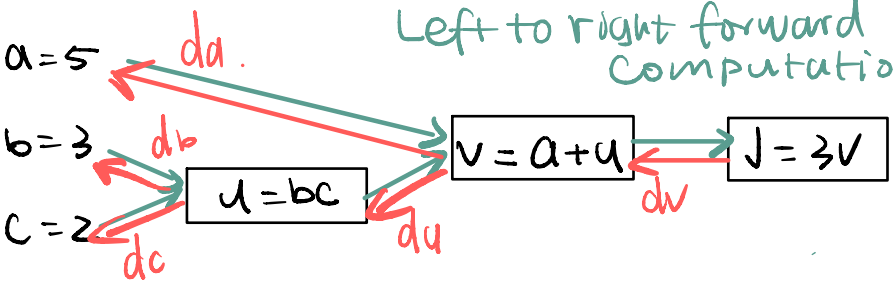
$$b := b - \alpha \frac{dJ(w,b)}{db}$$

Sometimes these are written as ∂, for partial derivative

Derivatives with a Computation Graph.

$J(a,b,c) = 3(a+bc)$

Computation Graph: Green lines. Left to right forward computation.



Red lines: right to left, backward derivatives.

da, db, dc, du, dv are abbs for partial derivatives $dJ/d*$

Multivariable calculus calculation:

$dv = 3$

$$du = \frac{dJ}{du} = \frac{dJ}{dv} \cdot \frac{dv}{du} = 3 \cdot 1 = 3$$

$$da = \frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da} = 3 \cdot 1 = 3$$

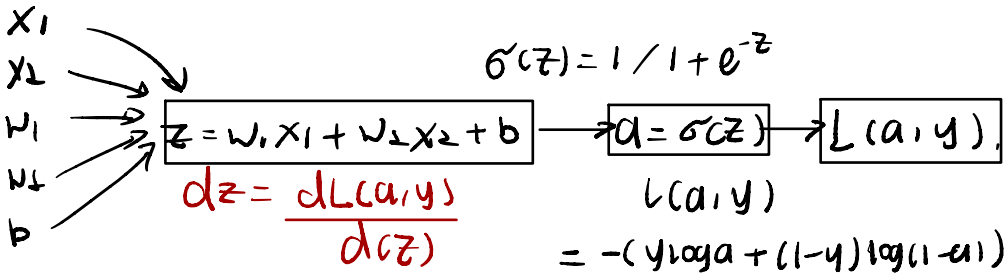
$$db = \frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 3c$$

$$dc = \frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc} = 3b$$

da, db, dc are also gradients (directions at which a, b, c can change J the most).

logistic regression derivatives.

Assume each input has 2 features, i.e. $x \in \mathbb{R}^2$.



$dz = \frac{dL(a,y)}{dz}$

$L(a,y) = -(y \log a + (1-y)\log(1-a))$

$dw_1 = x_1 dz$ $= \frac{dL(a,y)}{da} \cdot \frac{da}{dz}$

$dw_2 = x_2 dz$

$db = dz$

$da = \frac{dL(a,y)}{da}$

$= -\frac{y}{a} + \frac{1-y}{1-a}$

$\Rightarrow w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db.$

logistic regression on m examples

$$\frac{\partial}{\partial w_1} J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} L(a^{(i)}, y^{(i)})}_{dw_1^{(i)} - (a^{(i)}, y^{(i)})}$$

Formal Algorithm.

$J = 0$; $dw_1 = 0$; $dw_2 = 0$; $db = 0$

for $i = 1$ to m :

$z^{(i)} = w^T x^{(i)} + b$ loss Calculation

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$ Derivative Calculation.

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$; $dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$; $db \mathrel{+}= dz^{(i)}$

$J \mathrel{/}= m$

$dw_1 \mathrel{/}= m$; $dw_2 \mathrel{/}= m$; $db \mathrel{/}= m$

We have too many for loops which is not very efficient. Thus, we need Vectorization.